About the generalized LM-inverse and the Weighted Moore-Penrose inverse

Milan B. Tasić*, **Predrag S. Stanimirović**, **Selver H. Pepić**

University of Niš, Faculty of Sciences and Mathematics,

Višegradska 33, 18000 Niš, Serbia †

E-mail: `milan12t@ptt.rs`, `pecko@pmf.ni.ac.rs`, `p_selver@yahoo.com`

Abstract

The recursive method for computing the generalized LM- inverse of a constant rectangular matrix augmented by a column vector is proposed in [16, 17]. The corresponding algorithm for the sequential determination of the generalized LM-inverse is established in the present paper. We prove that the introduced algorithm for computing the generalized LM inverse and the algorithm for the computation of the weighted Moore-Penrose inverse developed by Wang in [23] are equivalent algorithms. Both of the algorithms are implemented in the present paper using the package MATHEMATICA. Several rational test matrices and randomly generated constant matrices are tested and the CPU time is compared and discussed.

AMS Subj. Class.: 15A09, 68W30.

Key words: Generalized inverses, LM-inverse, Weighted Moore-Penrose inverse, rational matrices, MATHEMATICA, Partitioning method.

1 Introduction

As usual, let $\mathbb{C}$ be the set of complex numbers, $\mathbb{C}^{m\times n}$ be the set of $m \times n$ complex matrices, and $\mathbb{C}_r^{m\times n} = \{X \in \mathbb{C}^{m\times n} : \text{rank}(X) = r\}$. For any matrix $A \in \mathbb{C}^{m\times n}$ and positive definite matrices M and N of the orders m and n respectively, consider the following equations in X, where $*$ denotes conjugate and transpose:

$$\begin{array}{llll} (1) & AXA = A & (2) & XAX = X \\ (3M) & (MAX)^* = MAX & (4N) & (NXA)^* = NXA. \end{array}$$

The matrix X satisfying equations (1), (2), (3M) and (4N) is called the weighted Moore-Penrose inverse of A, and it is denoted by $X = A^{\dagger}_{M,N}$. Especially, in the case $M = I_m$ and $N = I_n$, the matrix $X = A^{\dagger}_{M,N}$ becomes the Moore-Penrose inverse of A, and it is denoted by $X = A^{\dagger}$.

Various methods for computing the Moore-Penrose inverse of a matrix are known. The main methods are based on the Cayley-Hamilton theorem, the full-rank factorization and the singular value decomposition (see the example [1]). The Greville's *partitioning method*, introduced in [4], is one of the most efficient algorithms for computing the Moore-Penrose inverse. Two different proofs for the Greville's method were presented in [2, 24]. Udwadia and Kalaba gave an alternative and a simple constructive

*Corresponding author

†The authors gratefully acknowledge support from the research project 144011 of the Serbian Ministry of Science.

proof of Greville's formula in [19]. In [3] Fan and Kalaba determined the Moore-Penrose inverse of matrices using dynamic programming and the Belman's principle of optimality. Sivakumar in [12] used the Greville's formula for $A_k^\dagger = [A_{k-1} \,|a_k]^\dagger$ and just verified that it satisfies the four Penrose equations. This provides a proof of the Greville's method by the verification.

The Greville's algorithm is used in various computations, where its dominance is verified over various direct methods for the pseudoinverse computation. The computational experience presented in [7] is: "When applied to a square, fully populated, non-symmetric case, with independent columns, the Greville's algorithm was found that the approach can be up to 8 times faster than the conventional approach of using the SVD; rectangular cases are shown to yield similar levels of speed increase". The Greville's method has been used as a benchmark for the calculation of the pseudo-inverse.

Due to its computational dominance, this method has been extensively applied in many mathematical areas, such as statistical inference, filtering theory, linear estimation theory, optimization and more recently analytical dynamics [20] (see also [6]). An application in a direct approach for computing the gradient of the pseudo-inverse is presented in [7]. It has also found wide applications in database and the neural network computation [8]. In the paper [5], the sequential determination of the Moore-Penrose inverse by dynamic programming is applied to the diagnostic classification of electromyography signals.

There is a lot of extensions of the partitioning method. Wang in [23] generalized Greville's method to the weighted Moore-Penrose inverse. Also, the results in [23] are proved by using a new technique. Udwadia and Kalaba developed the recursive relations for the different types of generalized inverses [21, 22]. Finally, the Greville's recursive principle is generalized to various subsets of outer inverses and extended to the set of the one-variable rational and polynomial matrices in [15].

The algorithm for the computation of the Moore-Penrose inverse of the one-variable polynomial and/or rational matrix, based on the Greville's partitioning algorithm, was introduced in [13]. The extension of results from [13] to the set of the two-variable rational and polynomial matrices is introduced in the paper [10].

The Wang's partitioning method from [23], aimed in the computation of the weighted Moore-Penrose inverse, is extended to the set of the one-variable rational and polynomial matrices in the paper [14]. Also the efficient algorithm for computing the weighted Moore-Penrose inverse, appropriate for the polynomial matrices where only a few polynomial coefficients are nonzero, is established in [9].

In the paper [6] the authors derived a formula for the computation of the Moore-Penrose inverse of M^*M and obtained sufficient conditions for its nonnegativity, where $M = [A\,|\,a]$.

On the other side, there are a few articles which are interested in with computation of the generalized LM-inverse. The definition of the LM-inverse and the recursive algorithm of the Greville's type (for a matrix augmented by a column vector) are given in [16, 17]. The recursive relations in [16, 17] are proved by direct verification of the four conditions of the generalized LM-inverse. Also, these formulae are particularized to obtain recursive relations for the generalized L-inverse of a general matrix augmented by a column [17]. The recursive relations for the determination of the generalized Moore-Penrose M-inverse are derived in [18]. Separate relations for the situations when the rectangular matrix is augmented by a row vector and when such a matrix is augmented by a column vector are considered in [18]. The alternative proof for the determination of the generalized Moore-Penrose M-inverse of a matrix through the direct verification of the four properties of the Moore-Penrose M-inverse are presented in [11].

It is not difficult to verify that the conditions which characterize the generalized LM-inverse are equivalent with the corresponding equations characterizing the weighted Moore-Penrose inverse. Moreover, the matrix norms minimization used in (3) and (4) in the article [16] also characterizes the weighted Moore-Penrose inverse. Therefore, the generalized LM-inverse and the weighted Moore-Penrose inverse are identical. In the present paper we compare the corresponding algorithms. It is realistic to predict that algorithm for computing the weighted Moore-Penrose inverse from [23] and the algorithm for the computation of the generalized LM-inverse, introduced in the present paper and based on the results from [16, 17] are the same. Verification of this prediction is the main result of the present paper. Therefore, the present paper is continuation of the papers [9, 13, 14, 16, 17].

The structure of the present paper is as follows. In the second section we restate the representation of the generalized LM-inverse from [16, 17] as well as the representation and algorithm for computing the weighted Moore-Penrose from [23]. We also introduce an effective algorithm for construction of the generalized LM-inverse directly using its representation proposed in [16, 17]. In the third section we provide a proof that two algorithms from the second section are equivalent. Implementation of both algorithms and a few illustrative examples are presented.

2 Preliminaries and motivation

The recursive determination of the weighted Moore-Penrose inverse $A^{\dagger}_{M,N}$ is established in [23].

Let $A \in \mathbb{C}^{m\times n}$ and A_k be the submatrix of A consisting of its first k columns. For $k = 2, \dots, n$ the matrix A_k is partitioned as

$$A_k = [A_{k-1} \mid a_k], \tag{2.1}$$

where a_k is the k-th column of A.

Theorem 2.1 (G.R. Wang, Y.L. Chen [23]). *Let $A \in \mathbb{C}^{m\times n}$ and A_k be the submatrix of A consisting of its first k columns. For $k = 2, \dots, n$ the matrix A_k is partitioned as in (2.1), and the matrix $N_k \in \mathbb{C}^{k\times k}$ is the leading principal submatrix of N, and N_k is partitioned as*

$$N_k = \begin{bmatrix} N_{k-1} & l_k \\ l_k^* & n_{kk} \end{bmatrix}. \tag{2.2}$$

Let the matrices X_{k-1} and X_k be defined by

$$X_{k-1} = (A_{k-1})^{\dagger}_{M,N_{k-1}}, \quad X_k = (A_k)^{\dagger}_{M,N_k}, \tag{2.3}$$

the vectors d_k, c_k be defined by

$$\begin{aligned} d_k &= X_{k-1}a_k & (2.4)\\ c_k &= a_k - A_{k-1}d_k = (I - A_{k-1}X_{k-1})\,a_k. & (2.5)\end{aligned}$$

Then

$$X_k = \begin{bmatrix} X_{k-1} - (d_k + (I - X_{k-1}A_{k-1})\,N_{k-1}^{-1}l_k)b_k^* \\ b_k^* \end{bmatrix}, \tag{2.6}$$

where

$$b_k^* = \begin{cases} (c_k^* M c_k)^{-1}\, c_k^* M, & c_k \neq 0 \\ \delta_k^{-1}\,(d_k^* N_{k-1} - l_k^*)\,X_{k-1}, & c_k = 0, \end{cases} \tag{2.7}$$

and $\delta_k = n_{kk} + d_k^* N_{k-1} d_k - (d_k^* l_k + l_k^* d_k(s)) - l_k^*\,(I - X_{k-1}A_{k-1})\,N_{k-1}^{-1} l_k$.

According to the above theorem the next Algorithm 2.1 is introduced in [23].

Algorithm 2.1 Computing the weighted M-P inverse $A^{\dagger}_{M,N}$ using algorithm from [23].

```
Require: Let $A \in \mathbb{C}^{m\times n}$, $M$ and $N$ be p.d. matrices of the order $m$ and $n$ respectively.
 1: $A_1 = a_1$.
 2: if $a_1 = 0$, then
 3:     $X_1 = (a_1^* M a_1)^{-1} a_1^* M$;
 4: else
 5:     $X_1 = 0$.
 6: end if
 7: for $k = 2$ to $n$ do
 8:     $d_k = X_{k-1} a_k$,
 9:     $c_k = a_k - A_{k-1} d_k$,
10:     if $c_k \neq 0$, then
11:         $b_k^* = (c_k^* M c_k)^{-1} c_k^* M$, goto Step 16,
12:     else
13:         $\delta_k = n_{kk} + d_k^* N_{k-1} d_k - (d_k^* l_k + l_k^* d_k) - l_k^*(I - X_{k-1}A_{k-1})N_{k-1}^{-1} l_k$,
14:         $b_k^* = \delta_k^{-1}(d_k^* N_{k-1} - l_k^*) X_{k-1}$,
15:     end if
16:     $X_k = \begin{bmatrix} X_{k-1} - (d_k + (I - X_{k-1}A_{k-1})N_{k-1}^{-1} l_k) b_k^* \\ b_k^* \end{bmatrix}$.
17: end for
18: return $A^{\dagger}_{MN} = X_n$.
```

Also, the next auxiliary Algorithm 2.2, required in Algorithm 2.1 is stated in [23].

Algorithm 2.2 Computing the inverse matrix N^{-1}.

```
Require: Let $N_k = \begin{bmatrix} N_{k-1} & l_k \\ l_k^* & n_{kk} \end{bmatrix} \in \mathbb{C}^{k\times k}$ be the leading principal submatrix of p.d. matrix $N$.
1: $N_1^{-1} = n_{11}^{-1}$.
2: for $k = 2$ to $n$ do
3:     $g_{kk} = (n_{kk} - l_k^* N_{k-1}^{-1} l_k)^{-1}$,
4:     $f_k = -g_{kk} N_{k-1}^{-1} l_k$,
5:     $E_{k-1} = N_{k-1}^{-1} + g_{kk}^{-1} f_k f_k^*$,
6:     $N_k^{-1} = \begin{bmatrix} E_{k-1} & f_k \\ f_k^* & g_{kk} \end{bmatrix}$.
7: end for
8: return $N^{-1} = N_n^{-1}$.
```

The definition of the generalized LM-inverse is given in [16], and it is based on the usage of the linear equation $Ax = b$, where A is an $m \times n$ matrix, b is an m-vector and x is an n-vector. The matrix $A^{\dagger}_{LM}$ is such that the vector x, uniquely given by $x = A^{\dagger}_{LM} b$, minimizes both of the following two vector norms (conditions (3) and (4) from [16])

$$G = \| L^{1/2}(Ax - b) \|^2 = \| Ax - b \|_L^2,$$
$$H = \| M^{1/2} x \|^2 = \| x \|_M^2,$$

where L is an $m \times m$ symmetric positive-definite matrix and M is an $n \times n$ symmetric positive-definite matrix.

The recursive formulae for determining the generalized LM-inverse $A^\dagger_{L,M}$ of any given matrix A are introduced in [16, 17], and they are restated here for the sake of completeness.

Theorem 2.2 (F.E. Udwadia, P. Phohomsiri [16, 17]). *The generalized LM-inverse of any given matrix $B = [A \mid a] \in \mathbb{C}^{m\times n}$ is determined using the following recursive relations:*

$$B^\dagger_{L,M} = [A \mid a]^\dagger_{L,M} = \begin{cases} \begin{bmatrix} A^\dagger_{L,M\_} - A^\dagger_{L,M\_}\, a\, d^\dagger_L - p\, d^\dagger_L \\ d^\dagger_L \end{bmatrix}, & d = \left(I - AA^\dagger_{L,M\_}\right) a \neq 0; \\ \\ \begin{bmatrix} A^\dagger_{L,M\_} - A^\dagger_{L,M\_}\, ah - p\, h \\ h \end{bmatrix}, & d = \left(I - AA^\dagger_{L,M\_}\right) a = 0, \end{cases} \tag{2.8}$$

where A is an $m \times (n-1)$ matrix, a is a column vector of m components,

$$d^\dagger_L = \frac{d^T L}{d^T L d}, \ h = \frac{q^T M U}{q^T M q},$$
$$U = \begin{bmatrix} A^\dagger_{L,M\_} \\ 0_{1\times m} \end{bmatrix}, \ q = \begin{bmatrix} v + p \\ -1 \end{bmatrix}, \ v = A^\dagger_{L,M\_} a$$

and

$$p = \left(I - A^\dagger_{L,M\_} A\right) M_\_^{-1} \widetilde{m}.$$

Note that L is a symmetric positive definite $m \times m$ matrix, and

$$M = \begin{bmatrix} M\_ & \widetilde{m} \\ \\ \widetilde{m}^T & \bar{m} \end{bmatrix}, \tag{2.9}$$

where M is a symmetric positive-definite $n\times n$ matrix, $M\_$ is a symmetric positive-definite $(n-1)\times(n-1)$ matrix, $\widetilde{m}$ is a column vector of $n-1$ components, and $\bar{m}$ is a scalar.

Theorem 2.2 assumes in $B = [A \mid a]$ that the matrix B is obtained augmenting the matrix A by an appropriate column vector a. In the rest of the paper we assume that $B = [A \mid a]$ is just the partitioning (2.1): $B = A_k,\ A = A_{k-1},\ a = a_k$. Moreover, it is clear that the following notations immediately follows from Algorithm 2.1:

$$B^\dagger_{L,M} = X_k,\ A^\dagger = X_{k-1}.$$

Also, we use the following denotation for the matrix M defined in (2.9):

$$M = \begin{bmatrix} M_{k-1} & \widetilde{m}_k \\ \\ \widetilde{m}_k^T & m_{k,k} \end{bmatrix}, \tag{2.10}$$

Finally, the vector d corresponding to the first k columns of A is denoted by d_k.

According to the above Theorem 2.2 we introduce the next algorithm.

Algorithm 2.3 Computing the LM-inverse $A^{\dagger}_{LM}$ using the representation from [16].

```
Require: Let $A \in \mathbb{C}^{m\times n}$, $L$ and $M$ be p.d. matrices of order $m$ and $n$ respectively.
 1: $A_1 = a_1$.
 2: if $a_1 = 0$ then
 3:     $X_1 = (a_1^* L a_1)^{-1} a_1^* L$
 4: else
 5:     $X_1 = 0$.
 6: end if
 7: for $k = 2$ to $n$ do
 8:     $d_k = (I - A_{k-1}X_{k-1})\, a_k$,
 9:     $p = (I - X_{k-1}A_{k-1})\, M_{k-1}^{-1}\widetilde{m}_k$,
10:     if $d_k \neq 0$ then
11:         $b_k^* = \frac{d_k^* L}{d_k^* L d_k}$ goto Step 17
12:     else
13:         $q = \begin{bmatrix} X_{k-1}a_k + p \\ -1 \end{bmatrix}$,
14:         $U = \begin{bmatrix} X_{k-1} \\ 0 \end{bmatrix}$,
15:         $b_k^* = \frac{q^*}{q^* M_k q} M_k U$,
16:     end if
17:     $X_k = \begin{bmatrix} X_{k-1} - X_{k-1}a_k b_k^* - p\, b_k^* \\ b_k^* \end{bmatrix}$.
18: end for
19: return $A^{\dagger}_{L,M} = X_n$.
```

It is clear from restated definitions that the generalized LM-inverse is just the weighted Moore-Penrose inverse. Therefore, Algorithm 2.3 and algorithms 2.1, 2.2 together produce identical result - the weighted Moore-Penrose inverse of the given $m \times n$ matrix. In the next section we compare the described algorithms.

3 Comparison of algorithms

Theorem 3.1. *Algorithm 2.3 is equivalent to algorithms 2.1 and 2.2.*

Proof. In order to ensure unambiguous, during the proof we assume that the symbol W in a superscript denotes terms from the Wang's algorithm; similarly we use the convention that U, as a superscript, denotes terms from the Udwadia's algorithm, elsewhere it is necessary. Since the LM-inverse is just the weighted Moore-Penrose inverse, we conclude that the matrix M in Algorithm 2.1 is just the matrix L in Algorithm 2.3 and the matrix N in Algorithm 2.1 is analogous with the matrix M in Algorithm 2.3. Therefore, it is not necessarily to mark the matrices L, M, N and A by appropriate superscript.

We prove the theorem by verifying the equivalence of the outputs from the corresponding algorithmic steps of mentioned algorithms. The proof proceeds by the mathematical induction.

The proof for the case $k = 1$ in view of Step 3 in both algorithms is trivial. Assume that the statement is valid for the first $k - 1$ columns, i.e.

$$X^U_{k-1} = X^W_{k-1} = (A_{k-1})^{\dagger}_{M,N_{k-1}} = A^{\dagger}_{LM\_}. \tag{3.11}$$

Now we verify the inductive step. Wang used the matrix X_k in the form

$$X_k^W = \begin{bmatrix} X_{k-1}^W - \left(d_k^W + (I - X_{k-1}^W A_{k-1})\, N_{k-1}^{-1} l_k\right)(b_k^*)^W \\ (b_k^*)^W \end{bmatrix}, \tag{3.12}$$

while Udwadia observed two cases, as in (2.8).

If we denote with

$$(b_k^*)^U = \begin{cases} d_L^\dagger, & d_k^U \neq 0 \\ h, & d_k^U = 0, \end{cases} \tag{3.13}$$

then the equalities in (2.8) become

$$X_k^U = \begin{bmatrix} X_{k-1}^U - X_{k-1}^U a_k (b_k^*)^U - p\,(b_k^*)^U \\ (b_k^*)^U \end{bmatrix}. \tag{3.14}$$

Let us show that the output of Step 9 from Algorithm 2.1 is the same as the output of Step 8 from Algorithm 2.3:

$$c_k^W \equiv a_k - A_{k-1} d_k^W = a_k - A_{k-1} X_{k-1}^W a_k = \left(I - A_{k-1} X_{k-1}^W\right) a_k \equiv d_k^U.$$

Now we show that Step 11 from Algorithm 2.1 and Step 11 from Algorithm 2.3 are equivalent. As it is stated above $c_k^W = d_k^U$, so that in the case $c_k^W \neq 0$ we have

$$(b_k^*)^W = ((c_k^*)^W M c_k^W)^{-1} (c_k^*)^W M = ((d_k^*)^U L d_k^U)^{-1} (d_k^*)^U L = (b_k^*)^U.$$

In a similar way it can be verified that Step 14 from Algorithm 2.1 is equivalent to Step 15 from Algorithm 2.3. In the case $c_k^W = 0$ we can start from the statement in Step 15 from Algorithm 2.3.

$$(b_k^*)^U = (q^* M_k U)/(q^* M_k q).$$

From Step 9 of Algorithm 2.3 and the inductive hypothesis the following holds

$$p = \left(I - X_{k-1}^U A_{k-1}\right) N_{k-1}^{-1} l_k = \left(I - X_{k-1}^W A_{k-1}\right) N_{k-1}^{-1} l_k,$$

so that we derive the following:

$$\begin{aligned}
q^T M_k U &= \left[(X_{k-1}^U a_k + p)^* \quad | \quad -1\right] \begin{bmatrix} N_{k-1} & l_k \\ l_k^* & n_{kk} \end{bmatrix} \begin{bmatrix} X_{k-1}^U \\ 0 \end{bmatrix} \\
&= \left[(X_{k-1}^U a_k + p)^* N_{k-1} - l_k^* \quad | \quad (X_{k-1}^U a_k + p)^* l_k - n_{kk}\right] \begin{bmatrix} X_{k-1}^U \\ 0 \end{bmatrix} \\
&= (X_{k-1}^U a_k + p)^* N_{k-1} X_{k-1}^U - l_k^* X_{k-1}^U \ \{\text{since } \ X_{k-1}^U a_k = X_{k-1}^W a_k = d_k^W\} \\
&= (d_k^*)^W N_{k-1} X_{k-1}^W - l_k^* X_{k-1}^W + p^* N_{k-1} X_{k-1}^W \\
&= \left((d_k^*)^W N_{k-1} - l_k^*\right) X_{k-1}^W + \left(\left(I - X_{k-1}^W A_{k-1}\right) N_{k-1}^{-1} l_k\right)^* N_{k-1} X_{k-1}^W \\
&= \left((d_k^*)^W N_{k-1} - l_k^*\right) X_{k-1}^W + \left(l_k^* N_{k-1}^{-1} \left(I - X_{k-1}^W A_{k-1}\right)^*\right) N_{k-1} X_{k-1}^W \\
&= \left((d_k^*)^W N_{k-1} - l_k^*\right) X_{k-1}^W + l_k^* X_{k-1}^W - l_k^* N_{k-1}^{-1} \left(X_{k-1}^W A_{k-1}\right)^* N_{k-1} X_{k-1}^W.
\end{aligned}$$

Since N_{k-1} is the symmetric positive definite applying equality $(4N)$ together with (3.11) the following holds

$$\left(X_{k-1}^W A_{k-1}\right)^* N_{k-1} = \left(N_{k-1} X_{k-1}^W A_{k-1}\right)^* = N_{k-1} X_{k-1}^W A_{k-1},$$

and later

$$\begin{aligned} q^TM_kU &= \left((d_k^*)^WN_{k-1}-l_k^*\right)X_{k-1}^W+l_k^*X_{k-1}^W-l_k^*N_{k-1}^{-1}N_{k-1}X_{k-1}^WA_{k-1}X_{k-1}^W\\ &= \left((d_k^*)^WN_{k-1}-l_k^*\right)X_{k-1}^W+l_k^*\left(X_{k-1}^W-X_{k-1}^WA_{k-1}X_{k-1}^W\right)\\ &= \left((d_k^*)^WN_{k-1}-l_k^*\right)X_{k-1}^W+l_k^*\left(X_{k-1}^W-X_{k-1}^W\right)\\ &= \left((d_k^*)^WN_{k-1}-l_k^*\right)X_{k-1}^W. \end{aligned}$$

Moreover, we have

$$\begin{aligned} q^TM_kq &= \left[(X_{k-1}^Ua_k+p)^* \quad | \quad -1\right]\begin{bmatrix} N_{k-1} & l_k\\ l_k^* & n_{kk}\end{bmatrix}\begin{bmatrix} X_{k-1}^Ua_k+p\\ -1\end{bmatrix}\\ &= \left[(X_{k-1}^Ua_k+p)^*N_{k-1}-l_k^* \quad | \quad (X_{k-1}^Ua_k+p)^*l_k-n_{kk}\right]\begin{bmatrix} X_{k-1}^Ua_k+p\\ -1\end{bmatrix}\\ &= ((X_{k-1}^Ua_k+p)^*N_{k-1}-l_k^*)(X_{k-1}^Ua_k+p)-(X_{k-1}^Ua_k+p)^*l_k+n_{kk}\\ &= (X_{k-1}^Ua_k+p)^*N_{k-1}(X_{k-1}^Ua_k+p)-l_k^*(X_{k-1}^Ua_k+p)-(X_{k-1}^Ua_k+p)^*l_k+n_{kk}\\ &\quad\ (\text{since }\ \{X_{k-1}^Ua_k=X_{k-1}^Wa_k=d_k^W\})\\ &= (d_k^W+p)^*N_{k-1}(d_k^W+p)-l_k^*(d_k^W+p)-(d_k^W+p)^*l_k+n_{kk}\\ &= (d_k^*)^WN_{k-1}d_k^W-l_k^*d_k^W-(d_k^*)^Wl_k+n_{kk}-l_k^*p\\ &\quad\ +p^*N_{k-1}d_k^W+(d_k^*)^WN_{k-1}p+p^*N_{k-1}p-p^*l_k. \end{aligned}$$

Furthermore

$$p^*N_{k-1}d_k^W+(d_k^*)^WN_{k-1}p+p^*N_{k-1}p-p^*l_k=0. \tag{3.15}$$

First we show that $p^*N_{k-1}d_k^W=0$, as follows

$$\begin{aligned} p^*N_{k-1}d_k^W &= \left(\left(I-X_{k-1}^WA_{k-1}\right)N_{k-1}^{-1}l_k\right)^*N_{k-1}X_{k-1}^Wa_k\\ &= l_k^*N_{k-1}^{-1}\left(I-X_{k-1}^WA_{k-1}\right)^*N_{k-1}X_{k-1}^Wa_k\\ &= l_k^*N_{k-1}^{-1}\Big(N_{k-1}-\left(X_{k-1}^WA_{k-1}\right)^*N_{k-1}\Big)X_{k-1}^Wa_k\\ &= l_k^*N_{k-1}^{-1}\left(N_{k-1}-N_{k-1}X_{k-1}^WA_{k-1}\right)X_{k-1}^Wa_k\\ &= l_k^*N_{k-1}^{-1}N_{k-1}\left(I-X_{k-1}^WA_{k-1}\right)X_{k-1}^Wa_k\\ &= l_k^*\left(I-X_{k-1}^WA_{k-1}\right)X_{k-1}^Wa_k\\ &= l_k^*\left(X_{k-1}^W-X_{k-1}^WA_{k-1}X_{k-1}^W\right)a_k\\ &= l_k^*\left(X_{k-1}^W-X_{k-1}^W\right)a_k=0. \end{aligned}$$

Also, from the above equality, we have

$$(d_k^*)^WN_{k-1}p=(p^*N_{k-1}d_k^W)^*=0.$$

Finally, the last term of the sum in the left hand side of (3.15), is equal to $p^*N_{k-1}p-p^*l_k$, and it is also equal to zero:

$$\begin{aligned} p^*N_{k-1}p - p^*l_k &= p^*\left(N_{k-1}\left(I - X_{k-1}^W A_{k-1}\right)N_{k-1}^{-1}l_k - l_k\right) \\ &= p^*(-N_{k-1}X_{k-1}^W A_{k-1}N_{k-1}^{-1}\,l_k) \\ &= p^*\left(-(X_{k-1}^W A_{k-1})^*\,l_k\right) \\ &= \left(\left(I - X_{k-1}^W A_{k-1}\right)N_{k-1}^{-1}l_k\right)^*\left(-A_{k-1}^*(X_{k-1}^*)^W l_k\right) \\ &= -l_k^*N_{k-1}^{-1}A_{k-1}^*(X_{k-1}^*)^W l_k + l_k^*N_{k-1}^{-1}A_{k-1}^*(X_{k-1}^*)^W A_{k-1}^*(X_{k-1}^*)^W l_k \\ &= -l_k^*N_{k-1}^{-1}A_{k-1}^*(X_{k-1}^*)^W l_k + l_k^*N_{k-1}^{-1}A_{k-1}^*(X_{k-1}^*)^W l_k = 0. \end{aligned}$$

Continuing the transformation for $q^T M_k q$ we have

$$\begin{aligned} q^T M_k q &= (d_k^*)^W N_{k-1}d_k^W - l_k^*d_k^W - (d_k^*)^W l_k + n_{kk} - l_k^*\left(I - X_{k-1}^W A_{k-1}\right)N_{k-1}^{-1}l_k \\ &= \delta_k. \end{aligned}$$

Now we are able to continue the rest of our proof. According to the equality

$$(b_k^*)^U = (q^*M_kU)/(q^*M_kq),$$

Step 15 from Algorithm 2.3 produces the result

$$(b_k^*)^U = \delta_k^{-1}(d_k^*N_{k-1} - l_k^*)X_{k-1}^W,$$

which is identical to the output $(b_k^*)^W$, derived in Step 14 from Algorithm 2.1.

According to Step 16 of Algorithm 2.1 and Step 17 of Algorithm 2.3, the generalized LM-inverse and the weighted Moore-Penrose inverse of the first k columns of A are identical:

$$\begin{aligned} X_k^U &= \begin{bmatrix} X_{k-1}^U - \left(X_{k-1}^U a_k + p\right)(b_k^*)^U \\ (b_k^*)^U \end{bmatrix} \\ &= \begin{bmatrix} X_{k-1}^W - \left(d_k^W + \left(I - X_{k-1}^W A_{k-1}\right)N_{k-1}^{-1}l_k\right)(b_k^*)^W \\ (b_k^*)^W \end{bmatrix} \\ &= X_k^W. \end{aligned} \tag{3.16}$$

Finally, for the case $k = n$, it immediately follows that $A_{L,M}^\dagger = A_{M,N}^\dagger$, which means that the outputs from both algorithms are identical. □

4 Examples

In order to compare the algorithms from the second section it is necessary to use the precise implementation of the corresponding algorithms. Details concerning the implementation of the partitioning algorithm corresponding to the weighted Moore-Penrose inverse can be found in [14]. In order to compare the mentioned algorithms we developed a MATHEMATICA code for the implementation of Algorithm 2.3. We later tested results on different types of matrices. Since the language MATHEMATICA admits symbolic manipulation with data, developed implementations are immediately applicable to the rational and polynomial matrices.

Example 4.1. Consider the test matrix $A_{11\times 10}$ from [25], in the case $a = 1$

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 1 & 3 & 4 & 6 & 2 \\ 1 & 3 & 4 & 6 & 2 & 2 & 3 & 4 & 5 & 3 \\ 2 & 3 & 4 & 5 & 3 & 3 & 4 & 5 & 6 & 4 \\ 3 & 4 & 5 & 6 & 4 & 4 & 5 & 6 & 7 & 6 \\ 4 & 5 & 6 & 7 & 6 & 6 & 6 & 7 & 7 & 8 \\ 6 & 6 & 7 & 7 & 8 & 1 & 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 1 & 3 & 4 & 6 & 2 & 1 & 2 \\ 4 & 6 & 2 & 2 & 3 & 4 & 5 & 3 & 1 & 3 \\ 4 & 5 & 3 & 3 & 4 & 5 & 6 & 4 & 2 & 3 \\ 5 & 6 & 4 & 4 & 5 & 6 & 7 & 6 & 3 & 4 \\ 6 & 7 & 6 & 6 & 6 & 7 & 7 & 8 & 4 & 5 \end{bmatrix}_{11\times 10}$$

and randomly generated symmetric positive definite matrices $L_{10\times 10}$ and $M_{11\times 11}$:

$$L = \begin{bmatrix} 280 & -5 & -133 & -27 & -12 & -93 & -133 & -42 & 84 & 52 \\ -5 & 216 & -93 & -23 & 141 & -2 & -108 & -48 & 21 & 165 \\ -133 & -93 & 336 & 1 & 0 & 0 & 9 & -5 & 91 & 81 \\ -27 & -23 & 1 & 260 & -62 & -42 & -40 & 6 & 85 & -8 \\ -12 & 141 & 0 & -62 & 278 & -63 & -19 & -135 & 9 & 99 \\ -93 & -2 & 0 & -42 & -63 & 238 & 68 & -27 & -80 & -34 \\ -133 & -108 & 9 & -40 & -19 & 68 & 290 & 12 & -233 & -244 \\ -42 & -48 & -5 & 6 & -135 & -27 & 12 & 209 & -15 & -87 \\ 84 & 21 & 91 & 85 & 9 & -80 & -233 & -15 & 332 & 145 \\ 52 & 165 & 81 & -8 & 99 & -34 & -244 & -87 & 145 & 402 \end{bmatrix}_{10\times 10},$$

$$M = \begin{bmatrix} 452 & 91 & -186 & -97 & -161 & 68 & 28 & 16 & 151 & 41 & -65 \\ 91 & 413 & -74 & -119 & -317 & -41 & -12 & 67 & 180 & -53 & 54 \\ -186 & -74 & 497 & -136 & 78 & -208 & -175 & -120 & 9 & -99 & 6 \\ -97 & -119 & -136 & 371 & 157 & 154 & 129 & 29 & -102 & -16 & -96 \\ -161 & -317 & 78 & 157 & 444 & 28 & -39 & -201 & -165 & 3 & 43 \\ 68 & -41 & -208 & 154 & 28 & 509 & 52 & 55 & 90 & 179 & -81 \\ 28 & -12 & -175 & 129 & -39 & 52 & 454 & -38 & -157 & 145 & 22 \\ 16 & 67 & -120 & 29 & -201 & 55 & -38 & 408 & -9 & -16 & -100 \\ 151 & 180 & 9 & -102 & -165 & 90 & -157 & -9 & 257 & -32 & 14 \\ 41 & -53 & -99 & -16 & 3 & 179 & 145 & -16 & -32 & 376 & -15 \\ -65 & 54 & 6 & -96 & 43 & -81 & 22 & -100 & 14 & -15 & 339 \end{bmatrix}_{11\times 11}.$$

The generalized LM-inverse $A^{\dagger}_{L,M}$ from [16, 17] and the weighted Moore-Penrose inverse $A^{\dagger}_{M,N}$ from [23] are both equal to

$$\begin{bmatrix} 0.755 & -0.156 & -1.917 & 0.823 & 0.143 & 0.033 & 0.383 & -0.33 & 0.213 & -0.802 & 0.67 \\ -0.542 & -0.078 & 1.683 & -0.544 & -0.27 & 0.003 & -0.432 & 0.673 & 0.075 & 0.087 & -0.347 \\ 0.346 & -0.194 & -0.881 & 0.751 & -0.187 & 0.002 & 0.32 & -0.149 & 0.711 & -1.749 & 1.003 \\ -0.049 & 0.558 & -0.724 & 0.145 & 0.065 & 0.007 & 0.128 & -0.258 & -0.245 & 0.481 & -0.141 \\ -0.454 & -0.013 & 1.181 & -0.79 & 0.188 & 0.109 & -0.3 & 0.057 & -0.563 & 1.498 & -0.907 \\ -1.188 & -0.468 & 2.87 & -0.203 & -0.792 & -0.107 & 0.233 & 0.19 & 1.258 & -2.561 & 1.083 \\ 0.701 & 0.364 & -2.009 & 0.492 & 0.282 & 0.009 & 0.395 & -0.407 & -0.481 & 0.786 & -0.192 \\ 0.472 & 0.156 & -0.533 & -0.675 & 0.507 & -0.02 & -0.62 & -0.008 & -0.822 & 2.174 & -0.848 \\ -0.415 & -0.481 & 1.824 & -0.22 & -0.377 & 0.002 & 0.013 & 0.247 & 0.615 & -1.143 & 0.257 \\ 0.328 & 0.156 & -1.388 & 0.469 & 0.362 & -0.02 & 0.028 & -0.008 & -0.47 & 0.526 & -0.2 \end{bmatrix}.$$

The Moore-Penrose inverse can be generated in the case $L = M = \alpha I$, $M = N = \beta I$ [16], and it is equal to

$$A^{\dagger} = \begin{bmatrix} 0.294 & -0.169 & -1.511 & 1.415 & -0.391 & 0.041 & -0.04 & -0.26 & 0.454 & -0.264 & 0.23 \\ -0.067 & -0.074 & 1.227 & -1.064 & 0.23 & 0.001 & -0.192 & 0.649 & -0.103 & -0.191 & -0.102 \\ 0.227 & -0.179 & -0.692 & 0.705 & -0.214 & -0.009 & -0.254 & -0.238 & 1.514 & -1.319 & 0.449 \\ -0.165 & 0.547 & -0.657 & 0.376 & -0.116 & 0.015 & 0.179 & -0.196 & -0.456 & 0.531 & -0.104 \\ -0.297 & -0.01 & 1.035 & -0.972 & 0.359 & 0.108 & 0.238 & 0.044 & -1.065 & 0.938 & -0.366 \\ -0.008 & -0.474 & 1.663 & -1.319 & 0.352 & -0.103 & -0.428 & 0.224 & 1.83 & -1.844 & 0.414 \\ 0.062 & 0.368 & -1.349 & 1.081 & -0.33 & 0.006 & 0.426 & -0.434 & -0.442 & 0.712 & -0.156 \\ -0.081 & 0.158 & 0.029 & -0.144 & -0.034 & -0.021 & 0.087 & -0.019 & -1.499 & 1.448 & -0.138 \\ 0.195 & -0.484 & 1.198 & -0.791 & 0.211 & 0.005 & -0.19 & 0.268 & 0.765 & -0.906 & 0.049 \\ -0.081 & 0.158 & -0.971 & 0.856 & -0.034 & -0.021 & 0.087 & -0.019 & -0.499 & 0.448 & -0.138 \end{bmatrix}.$$

Example 4.2. Consider the one variable test matrix

$$A = \begin{bmatrix}
x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x^2 & x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x^3 & x^2 & x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x^4 & x^3 & x^2 & x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
x^5 & x^4 & x^3 & x^2 & x & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 & 0 & 0 & 0 & 0 & 0 \\
x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 & 0 & 0 & 0 & 0 \\
x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 & 0 & 0 & 0 \\
x^9 & x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 & 0 & 0 \\
x^{10} & x^9 & x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 & 0 \\
x^{11} & x^{10} & x^9 & x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 \\
x^{12} & x^{11} & x^{10} & x^9 & x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x
\end{bmatrix}$$

proposed in [25] and L (resp. M) and M (resp. N) as the identity matrices of the appropriate dimensions. Both of the considered algorithms produce the following Moore-Penrose inverse:

$$A^{\dagger} = \begin{bmatrix}
\frac{x}{x^2+1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{x^2+1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -x & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -x & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -x & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -x & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x & \frac{1}{x^2+1} & \frac{x}{x^2+1}
\end{bmatrix}.$$

Example 4.3. The CPU time needed for the computation of the generalized LM-inverse and the weighted Moore-Penrose inverse (according to Algorithm 2.3 and the algorithms 2.1, 2.2, respectively) is compared in the next table. The testing is done on the local machine with the following performances: Windows edition: Windows Home Edition; Processor: Intel(R) Celeron(R) M CPU @ 1.6GHz; Memory (RAM): 512 MB; System type: 32-bit Operating System; Software: MATHEMATICA 5.2. Also, in the table there are arranged results obtained on the set of randomly generated test matrices $A_{m\times n}$ and randomly generated symmetric positive definite matrices $L_{m\times m}$ (resp. $M_{m\times m}$) and $M_{n\times n}$ (resp. $N_{n\times n}$):

$m \times n$	degree	*Algorithm* 2.1 $A^{\dagger}_{MN}$	*Algorithm* 2.3 $A^{\dagger}_{LM}$
5x6	1	2.265 Seconds	2.235 Seconds
5x6	2	4.078	4.063
6x4	5	9.969	9.625
6x4	10	23.328	21.11
10x11	1	104.109	103.484
10x11	2	192.734	186.297
11x10	1	133.469	130.125
11x10	2	261.359	227.047

Table 1. The comparison in the efficiency on the set of randomly generated test matrices

According to the above Table 1 results it is evident that Algorithm 2.3 produces negligibly better performances with respect to Algorithm 2.1 for all test cases. This fact is in accordance with the verified equivalence between the algorithms.

5 Conclusions

Our primary idea is to show that the computational method for the generalized LM-inverse from [16, 17] and the algorithms for the computation of the weighted Moore-Penrose inverse from [23] are equivalent. The effective algorithm for the computation of the generalized LM-inverse is introduced here. Equivalence of the considered algorithms is proved in the third section by verifying the equivalence of outputs generated by corresponding algorithmic steps. This paper not only compares the corresponding algorithms but also compares the performance of two approaches of finding the Moore-Penrose inverse. In order to compare the efficiency of corresponding algorithms we developed their implementations in the programming language MATHEMATICA.

6 Appendix

Several auxiliary procedures implemented in MATHEMATICA are described at the beginning.

```
TakeCol[A_, k_] := Transpose[Take[Transpose[A], {k}]];
TakeCols[A_, k_] := Transpose[Take[Transpose[A], k]];
TakeElement[A_, i_, j_] := TakeCol[Take[A, {i}], j];
TakeElements[A_, i_, j_] := TakeCols[Take[A, i], j];
T[A_] := Transpose[A];
RandomPoly[n_, prob1_, prob2_, var_] := Module[{S, i, l},
   If [Random[Real, {0, 1}] > prob1, Return[0]];
   S = 0;
   Do[ If [Random[Real, {0, 1}] < prob2, l = 1, l = 0];
      S = S + var^i*Random[Integer, {-10, 10}]*l, {i, 0, n}];
   If [(S == 0) && (prob1 >= 1), S = S + 1];
   Return[S]];
RandomMatrix[m_, n_, deg_, prob1_, prob2_, var_] := Module[{A, i, j},
   A = Table[0, {i,1,m}, {j,1,n}];
   Do[Do[A[[i, j]] = RandomPoly[deg, prob1, prob2, var], {i,1,m}], {j,1,n}];
   Return[A]];
```

Implementation of Algorithm 2.3 is given by the following function.

```
ALMW[A_, M_, N_] :=
   Module[{I, m, n, d, c, tmp, tmp1, u, q, p, a, A1, l, X, N1, N2, n11},
    a = TakeCol[A, 1];   {m, n} = Dimensions[A];
    If [Together[a] === 0*a,
     X = T[a],
     X = Inverse[T[a].M.a].T[a].M ];
    A1 = {};
  Do[I = IdentityMatrix[i - 1];
     a = TakeCol[A, i]; A1 = TakeCols[A, i - 1];
     n11 = TakeElement[N, i, i];   N1 = TakeElements[N, i - 1, i - 1];
     N2 = TakeElements[N, i, i];    l = T[TakeCols[Take[N, {i}], i - 1]];
     d = (IdentityMatrix[m] - A1.X).a // Together;
     p = (I - X.A1).Inverse[N1].l // Together;
     If [Together[d] =!= 0*d,
      b = Inverse[T[d].M.d].T[d].M; b = T[b] // Together,
      tmp = (X.a + p) // Together; q = Join[tmp, {{-1}}];
      u = Append[X, Table[0, {j, m}]];
      b = (T[q].N2.u) Inverse[T[q].N2.q][[1]];
      b = T[b] // Together;
      ];
     tmp1 = X - (X.a - p).T[b] // Together;
     X = Together[Join[tmp1, T[b]]];
    , {i, 2, n}];
    Return[X // MatrixForm]];
```

Implementation of Algorithm 2.1 is obtained by slightly adopting the MATHEMATICA code described in [14].

```
AWang[A_, M_, N_] :=
 Module[{I, m, n, d, c, a, A1, l, del, X, tmp, tmp1, N1, n11},
   a = TakeCol[A, 1];
   {m, n} = Dimensions[A];
   If [Together[a] === 0*a,
     X = T[a],
     X = Inverse[T[a].M.a].T[a].M ]; A1 = {};
    Do[ I = IdentityMatrix[i - 1];
     a = TakeCol[A, i];  A1 = TakeCols[A, i - 1];
     d = X.a;    c = Together[a - A1.d];
     n11 = TakeElement[N, i, i];
     N1 = TakeElements[N, i - 1, i - 1];
     l = T[TakeCols[Take[N, {i}], i - 1]];
     tmp = T[d].N1 - T[l];
     tmp1 = (I - X.A1).Inverse[N1].l // Together;
     If [Together[c] =!= 0*c,
        b = Inverse[T[c].M.c].T[c].M;
        b = T[b] // Together,
        del = n11 + T[d].N1.d - (T[d].l + T[l].d) - T[l].tmp1 // Together;
        b = T[Inverse[del].tmp.X] // Together;
       ];
      X = Together[Join[X - (d + tmp1).T[b], T[b]]];
       , {i, 2, n}];
     Return[X]];
```

References

[1] A. Ben-Israel and T.N.E. Greville, *Generalized inverses: theory and applications*, Second Ed., Springer, 2003.

[2] S.L. Campbell and C.D. Meyer, Jr., *Generalized inverses of linear transformations*, London, Pitman, 1979.

[3] Y. Fan a and R. Kalaba, *Dynamic programming and pseudo-inverses*, Appl. Math. Comput. **139** (2003), 323-342.

[4] T.N.E. Greville, *Some applications of the pseudo-inverse of matrix* SIAM Rev. **3** (1960), 15–22.

[5] C. Itiki, *Dynamic programming and diagnostic classification*, J. Optim. Theory Appl. **127** (2005), 579-586.

[6] T. Kurmayya and K.C. Sivakumar, *Moore-Penrose inverse of a Gram matrix and its nonnegativity*, J. Optim. Theory Appl. **139** (2008), 201-207.

[7] J.B. Layton, *Efficient direct computation of the pseudo-inverse and its gradient*, Internat. J. Numer. Methods Engrg. **40** (1997), 4211–4223.

[8] S. Mohideen and V. Cherkassky, *On recursive calculation of the generalized inverse of a matrix*, ACM Trans. Math. Software **17** (1991), 130-147.

[9] M.D. Petković, P.S. Stanimirović and M.B. Tasić, *Effective partitioning method for computing weighted MoorePenrose inverse*, Comput. Math. Appl. **55** (2008), 1720-1734.

[10] M.D. Petković and P.S. Stanimirović, *Symbolic computation of the Moore-Penrose inverse using partitioning method*, Int. J. Comput. Math. **82** (2005), 355–367.

[11] P. Phohomsiri, B. Han, *An alternative proof for the recursive formulae for computing the MoorePenrose M-inverse of a matrix*, Appl. Math. Comput. **174** (2006), 81-97.

[12] K. C. Sivakumar, *Proof by verification of the Greville/Udwadia/Kalaba formula for the Moore-Penrose inverse of a matrix*, J. Optim. Theory Appl. **131** (2006), 307-311.

[13] P.S. Stanimirović and M.B. Tasić, *Partitioning method for rational and polynomial matrices*, Appl. Math. Comput. **155** (2004), 137–163.

[14] M.B. Tasić, P.S. Stanimirović, M.D. Petković, *Symbolic computation of weighted Moore-Penrose inverse using partitioning method*, Appl. Math. Comput. **189** (2007), 615–640.

[15] M.B. Tasić, P.S. Stanimirović, *Symbolic and recursive computation of different types of generalized inverses*, Appl. Math. Comput. **199** (2008), 349–367.

[16] F.E. Udwadia and P.Phohomsiri, *Generalized LM-inverse of a matrix augmented by a column vector*, Appl. Math. Comput. **190** (2007), 999–1006.

[17] F.E. Udwadia and P.Phohomsiri, *Recursive Formulas for Generalized LM-Inverse of a Matrix*, J. Optim. Theory Appl. **131** (2007), 1–16.

[18] F. E. Udwadia and P. Phohomsiri, *Recursive Determination of the Generalized MoorePenrose M-Inverse of a Matrix*, J. Optim. Theory Appl. **127** (2005), 639-663.

[19] F.E. Udwadia and R.E. Kalaba, *An Alternative Proof for Greville's Formula*, J. Optim. Theory Appl. **94** (1997), 23-28.

[20] F.E. Udwadia and R.E. Kalaba, *Analytical Dynamics: A New Approach*, Cambridge University Press, Cambridge, England, 1996.

[21] F.E. Udwadia and R.E. Kalaba, *A Unified Approach for the Recursive Determination of Generalized Inverses*, Comput. Math. Appl. **37** (1999), 125-130.

[22] F.E. Udwadia and R.E. Kalaba, *General forms for the Recursive Determination of Generalized Inverses: Unified approach*, J. Optim. Theory Appl. **101** (1999), 509–521.

[23] G.R. Wang and Y.L.Chen, *A recursive algorithm for computing the weighted Moore-Penrose inverse* $A^{\dagger}_{MN}$, J. Comput. Math. **4** (1986), 74–85.

[24] G.R. Wang, *A new proof of Greville's method for computing the weighted M-P inverse*, J. Shangai Teach. Univ., Nat. Sci. Ed. **3** (1985), 32–38.

[25] G. Zielke, *Report on test matrices for generalized inverses*, Computing **36** (1986), 105–162.